Security Vulnerability Assessment for an E-Commerce Platform

ENPM665 Section 0101 – Cloud Security
Midterm Group Project
Group 14

Name	Directory ID	UID
Viraj Sanjay Kurhade	vkurhade	122010192
Kalpesh Bharat Parmar	kalpesh	122046788
Chandana Charitha Peddinti	charitha	121288875
Ashley Nichole Rose	anrose	121381811
Vineet Agarwal	vineet54	121386052
Fahad Shaker	fshaker	119529003

This report evaluates security weaknesses in an AWS-based e-commerce application, identifying misconfigurations across IAM, data, and network layers, and recommending measures to strengthen confidentiality, integrity, and availability.

Table of Contents

List of Figures

Executive Summary

Several serious flaws in data security, virtual machine setups, network controls, logging and monitoring, disaster recovery, and application-level vulnerabilities were found during the security evaluation of the Juice Shop application hosted on AWS. SQL injection and XSS vulnerabilities, hardcoded and weak database credentials, unencrypted data in transit and at rest, out-of-date software components, publicly exposed ports, inadequate IAM governance, inadequate logging and threat detection, and a single point of failure in the disaster recovery setup are some of the main findings. When taken as a whole, these problems present serious threats to availability, confidentiality, and integrity, including possible illegal access, data exfiltration, system compromise, and noncompliance with regulations.

Strong data protection techniques like encryption, secret management, and parameterized queries should be put in place to reduce these risks. Network segmentation, access controls, and secure EC2 instance configuration should also be enforced. Centralized logging, continuous monitoring, and threat detection through CloudTrail, GuardDuty, and VPC Flow Logs should be enabled. Finally, a prepared disaster recovery plan with automated backups and multi-AZ (Multiple Availability Zones) redundancy should be established. Regular security testing, secure authentication and access management, and appropriate input validation are further ways to reinforce application-level controls. These high-level actions will guarantee quick detection and response, improve visibility, strengthen overall security posture, and lessen the possibility of operational, financial, and reputational damages.

Access Control and IAM Assessment

Purpose

In this article, the OWASP Juice Shop's web application and cloud infrastructure will be evaluated for vulnerabilities related to identity and access management (IAM) and access controls. The analysis will discuss how permissions could be tightened to mitigate risk in addition to theoretical improvements and best practices for IAM.

IAM & Access Control Vulnerability Assessment

SSL certificate misconfiguration

The website's SSL certificates were not configured correctly, rendering HTTPS protocol inoperable.

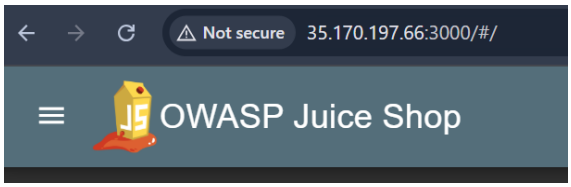

Figure 1: Website Using Unsecure HTTP Protocol

Impact:
Customers and partners could not securely access the website, exposing data to potential interception and damaging trust, compliance posture, and brand reputation.

Remediation:
Reconfigure and validate SSL certificates to enable HTTPS, ensure automatic renewal, and implement continuous monitoring to prevent recurrence.

Exposed PII (Email IDs)

The application shows exposed user emails (also used as usernames) in item reviews. Personally identifiable information (PII), such as a user email, should not be publicly exposed in this manner - especially since emails are used as login credentials. In this example, the email for the admin user (admin@juicesh.op) was compromised.

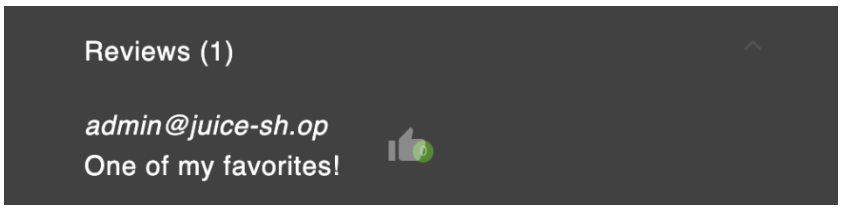

Figure 2: Entire Email Ids (PII) Are Directly Visible

Impact:
Public exposure of user emails increases the risk of phishing, credential stuffing, and privacy breaches, undermining user trust, and compliance with data protection standards.

Remediation:
Mask or anonymize user identifiers in public views and update the application to use non-PII display names; review and enforce data privacy controls.

Path Disclosure

URL resource paths were found in web application source code using browser developer tools. Admin consoles of this nature should never be made publicly accessible on the same domains accessed by ordinary user clients.

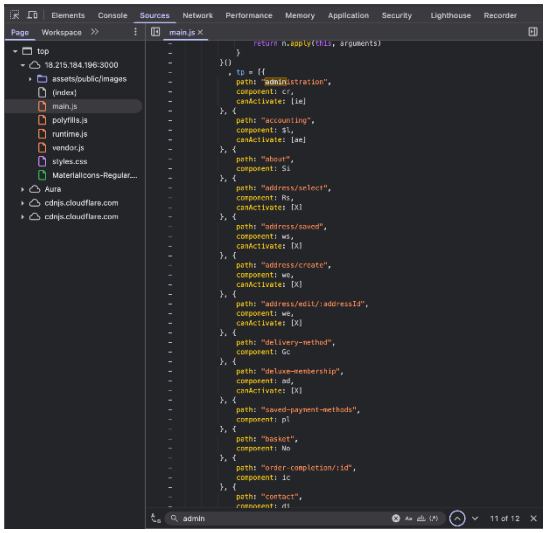

Figure 3: Path Disclosure For Webpages

Impact:
Exposing admin URLs increases the risk of unauthorized access, privilege escalation, and compromise of critical systems.

Remediation:
Relocate admin consoles to restricted, non-public domains with strong authentication and network access controls; remove sensitive paths from client-side code. Consider migrating this service to another instance accessed by internal DNS routes or VPN only.

Weak Credentials

After discovery of admin email, the admin password was cracked using brute force. password: admin123.

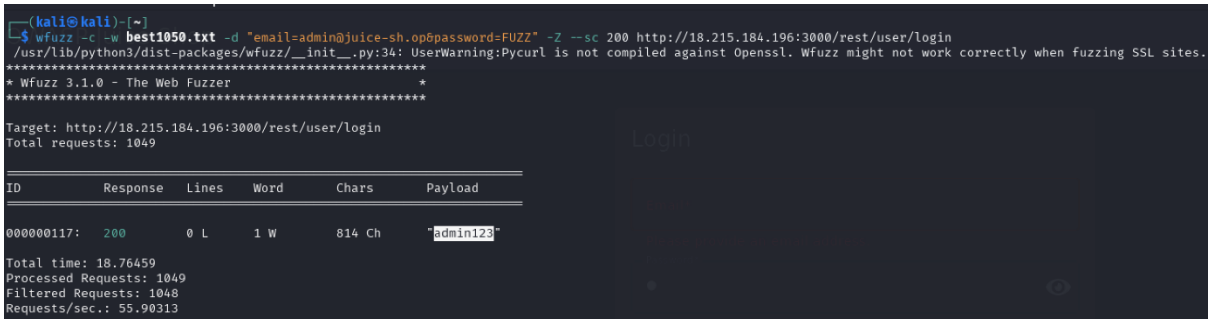

Figure 4: Cracking Password For Admin User

Impact:
Weak admin credentials allowed unauthorized access, risking full system compromise, data theft, and operational disruption.

Remediation:
To mitigate this risk, consider using a password manager, such as 1Password, to generate more secure credentials containing a longer string with a mix of alphanumeric and special characters. Consider using AWS Secrets Manager if these credentials need to be accessed programmatically. Implement account lockout and monitoring to detect brute-force attempts.

Exposed FTP web-directory

An unencrypted FTP web-directory was publicly exposed through a webpage.

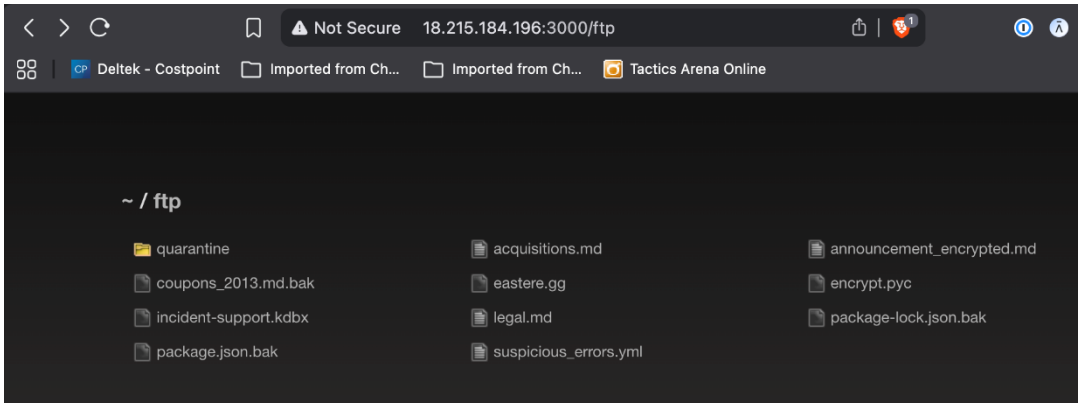

Figure 5: Exposed FTP Web-Directory

Impact:
Exposes sensitive files to unauthorized access, data theft, or malware injection.

Remediation:
Remove public access, disable or secure FTP with authentication and encryption, and restrict access to authorized users only.

Further Mitigation Steps

Most of the following mitigation steps will leverage the principle of least privilege, which is a security concept that states users and applications should only have access to the information and resources necessary to perform their tasks, minimizing potential risks and vulnerabilities. This approach helps protect systems from unauthorized access and reduces the impact of security breaches.

1. The database containing user credentials should be separated into a separate RDS instance in a private subnet, preferably a multi-AZ cluster for high availability and automated maintenance. Since only users with a need to see this data should have access to this database, create specific IAM roles and assigning it only to users with the database managers and administrators IAM group only.
2. AWS Secrets Manager is used to store credentials for programmatic use (such as admin and database credentials), strictly restrict access to the secrets manager.
3. (SAML Federation) should be leveraged for authentication instead of creating an IAM user for each individual user. Benefits of SAML federation include enhanced security through signed

and encrypted requests, improved user experience with single sign-on (SSO), and better compatibility with various identity providers.

4. For the exposed FTP server, move the content in this server to private S3 buckets to mitigate risk. Organize files into buckets appropriately. By organizing the files in this manner, S3 bucket policies can be configured such that only users with the appropriate IAM role can view the appropriate content.

Key Takeaways

The assessment revealed several gaps in how user identities and access permissions are managed. Exposed emails, weak passwords, and an openly accessible admin console make the system vulnerable to attacks and data leaks. Strengthening password policies, encrypting all traffic, and limiting admin access to internal networks would greatly reduce these risks. Moving sensitive data to private infrastructure and using tools like AWS Secrets Manager or SAML for authentication can further improve security. Overall, tightening IAM controls and following the principle of least privilege will help OWASP Juice Shop build a safer and more reliable system for both users and administrators.

Vulnerability Assessment Report

Introduction

This assessment revealed multiple critical security vulnerabilities in the application, this includes vulnerabilities in authentication, access control, data integrity, server configuration, and input validation. Issues like SQL injection, exposed confidential files, hidden admin pages, unencoded URLs, client-side script execution, and API misuse may enable external attackers to bypass authentication to gain access and impersonate users, manipulate the data. This highlights the security flaws in design as well as implementation.

Web-Application Vulnerability Assessment Report

Broken Authentication – SQL injection

The login form is vulnerable to SQL injection — specially crafted inputs like username'-- or ' OR 1=1 -- bypass authentication and allow logging in as any user (including admin) without a password.

Figure 6: User Login By SQL Injecting The Login Page

Figure 7: Admin Login By SQL Injecting The Login Page

Impact:
An attacker can bypass authentication, access admin account. This can compromise sensitive data, and may lead to escalated breach, damaging reputation of the organization and causing regulatory fines.

Remediation:
Queries should be parameterized and user inputs should be validated to prevent SQL injections. Admin accounts should be enforced with multi-factor authentication (MFA). Proper security monitoring should be there for high-privilege accounts.

Broken Access Control

A hidden admin page was discovered in the client-side code and it is accessible directly. This page should not be reachable without proper authorization. Admin user's credentials were easily obtained by SQL Injection without any username or password.

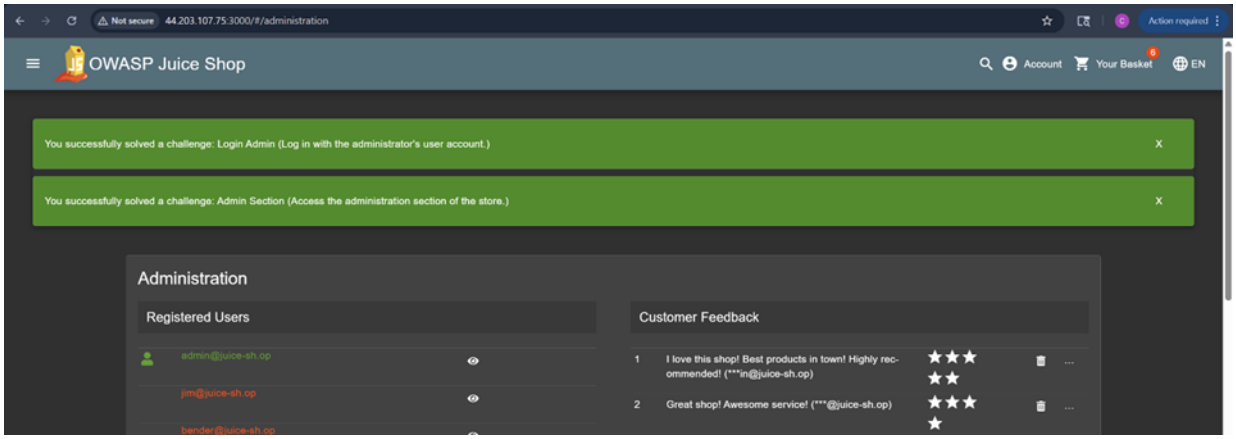

Figure 8: Administration Page

Impact (C-Suite Level):
Exposed sensitive webpage to admin panel was found. This allowed to visibility to customer feedback and even access to delete them. This poses significant business risk, including potential data breaches, regulatory fines, and damage to reputation of the organization.

Remediation (C-Suite Level):
Ensure that sensitive webpages are not exposed to the public. User inputs should be validated for SQL injections. Admin accounts should be enforced with multi-factor authentication (MFA). Proper security monitoring and logging should be enabled for such webpages.

Injection - XSS

Search input was vulnerable to client-side script injection. Here unvalidated input allowed to execute JavaScript code to run from the browser of victim machine.

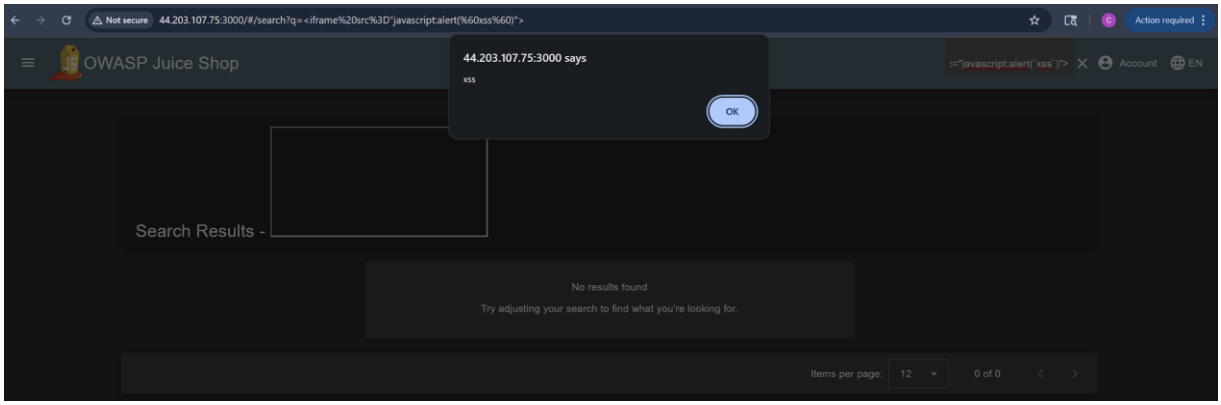

Figure 9: Successful XSS Injection

Impact:
XSS vulnerability in Document Object Model can allow attackers allowed to execute JavaScript code

to run from the browser of victim machine. As a result, sensitive data, including session cookies, login passwords, and personal information can be stolen. Additionally, it can allow users to be redirected to harmful websites, take over their accounts.

Remediations:
User inputs should be validated and sanitized before inserting them. strong Content Security Policy (CSP) should be implemented to disable script execution. Dynamically generating HTML with user-controlled data should be avoided and replaced with frameworks such as React or Angular, which automatically escape output.

Insecure Design

An image URL containing # was not properly encoded, allowing direct access when replaced with %23, indicating improper input validation and missing URL encoding.

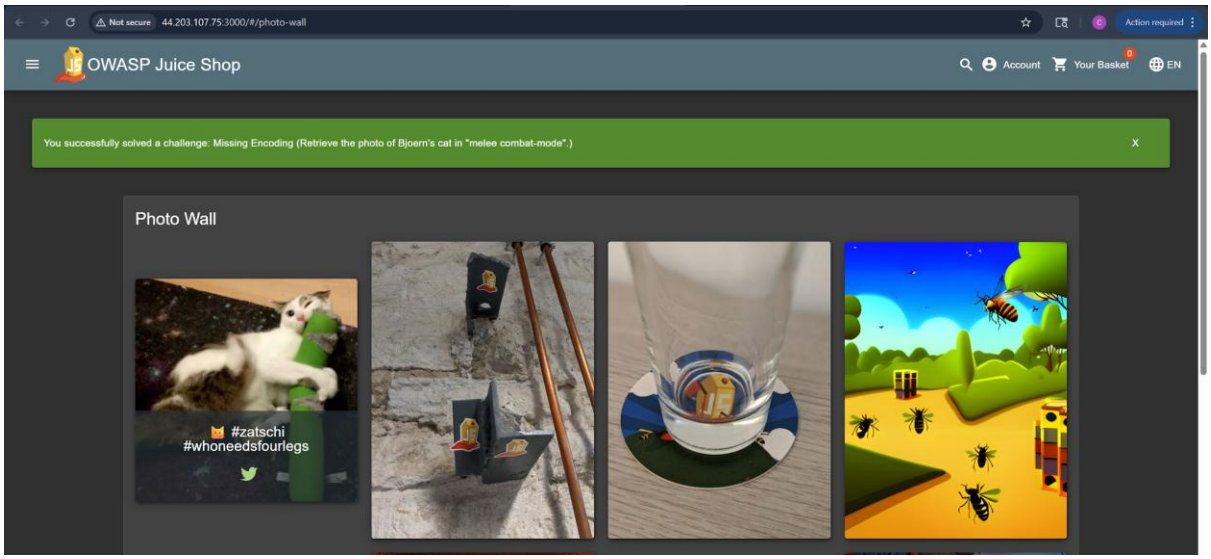

Figure 10: Improper URL Encoding

Impact:
Improper encoding can be exploited to bypass filters, or manipulate URLs to perform XSS or other injection attacks, and access restricted resources. The attackers can exploit such vulnerability to gain unauthorized access and view sensitive content.

Remediation:
URLS and user inputs should be properly encoded and validated before processing them. Regularly test for unencoded special characters that could lead to security bypasses.

Security Misconfiguration: Sensitive Data Leakage

A publicly accessible terms file revealed a directory path that exposed a confidential file stored unencrypted on the web server.

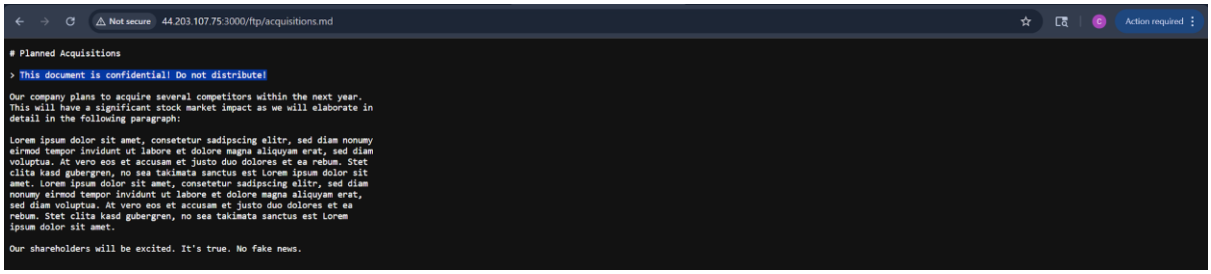

Figure 11: Publicly Accessible Confidential File

Impact:

Such exposed and unencrypted confidential files lead to immediate data breach. This can escalate to unauthorized access, regulatory or privacy violations, reputational damage to organization. Such data leak can be used by competitor organizations to gain business advantage.

Remediation:

Remove such files from publicly accessible web-directories. Use server-side authorization checks for hiding directory contents from client-side files. Encrypt sensitive files at rest and in transit.

Identification and Authentication Failures

The account creation page validates the repeat password only once. First password can be changed after validation and inconsistent passwords can be submitted, this bypass intended checks. bypassing intended checks. This affects the integrity of the authentication process.

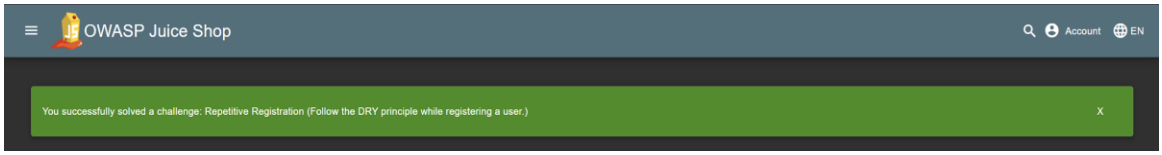

Figure 12: Inconsistent Password Validation

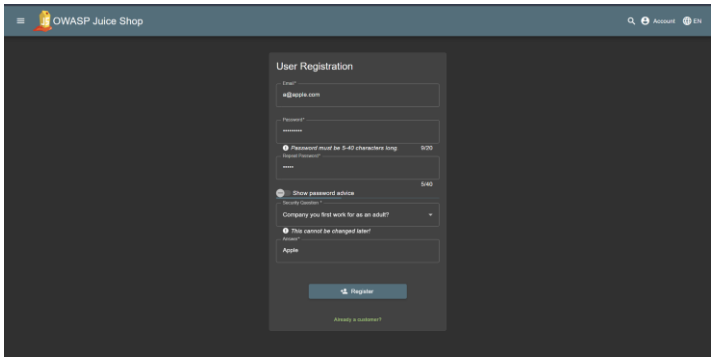

Figure 13: Inconsistent Password Validation in User Registration

Impact:

This flaw can lead to user confusion causing them to mistype passwords. This increases failed logins and increases the need and cost of user support. It raises questions on integrity and validation of the entire authentication process.

Remediation:

Implement server-side validation to confirm that passwords match while creating the user profile.

Enforce consistent input checks in multiple layers. Rigorously test such functionalities, preferably using automated as well as manual testing.

Software and Data Integrity Failures

The API accepts client-supplied usernames, allowing an attacker to post reviews on behalf of other users without authorization.

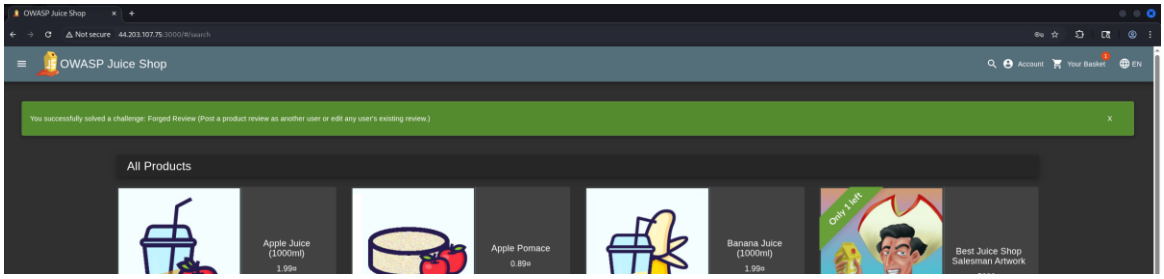

Figure 14: Forging Reviews

Impact:
This vulnerability can seriously damage platform trust and credibility. Attackers can manipulate reviews to mislead customers, harm brand reputation, manipulate customer emotions towards a product.

Remediation:
All requests should be validated on server-side and linked to authenticated user's session. Identities should not be delivered from Client inputs. Implement strict access controls, signed or tokenized requests.

Overall Impact

These vulnerabilities put the organization under significant risk. Exploitation of these can lead to full system compromise, sensitive data exposure, data tampering, regulatory violations, financial loss, and severe damage to organization's reputation. This also raises questions over trust and customer confidence on the organization.

Remediation Overview

For mitigation of these risks, strict server-side validation should be enforced, database queries should be parameterized, authentication and access controls should be secured, encode and sanitize all user inputs, protect sensitive data, and implement robust logging and monitoring. Regularly test the application using both manual and automated testing. Also, review code and website design and implementation at regular intervals.

Key Takeaways

These risks should be remediated immediately to protect the infrastructure. It is essential to maintain customer trust and ensure regulatory compliance. To reduce risk exposure, implement strengthened security across authentication, access control, data integrity, and input validation. This will safeguard the organization's reputation and operations.

Data Security Assessment Report

Introduction

This Data Security Assessment focuses on ensuring the confidentiality, integrity, and availability (CIA) of data processed and stored within the OWASP Juice Shop application deployed through an AWS CloudFormation stack. The primary objective is to evaluate data protection mechanisms implemented across the EC2 instance, MariaDB database, and application configuration layers, including authentication, encryption, and secret management.

The assessment identified several critical and high-severity vulnerabilities that exposed sensitive data, including database credentials, configuration secrets, and user information. These findings highlight the key risks associated with violating data protection best practices and cloud security standards.

Summary of Findings

Vulnerability Name	Severity
SQL Injection	Critical
Hardcoded / Leaked DB Credentials (user-data)	Critical
Weak Database Password & Missing Secure Initialization	Critical
No Encryption at Rest	Critical
No Encryption in Transit	Critical
Insecure Secrets Management	High
Database Auditing & Logging Disabled	Moderate

SQL Injection

Untrusted input is concatenated into SQL queries in both **product search** and **login** flows, enabling injection to enumerate DB contents and bypass authentication.

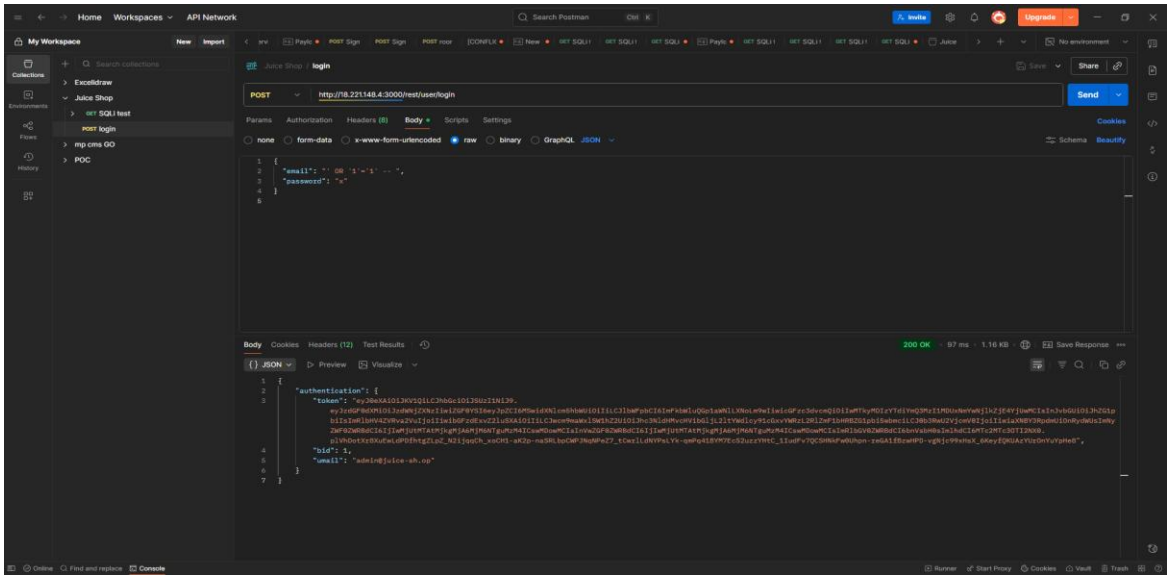

Figure 15: SQL Injection

Impact:

A successful database compromise would expose sensitive user and business data, allowing attackers to read, alter, or delete critical records. This could disrupt operations, cause service outages, and lead to significant confidentiality, integrity, and availability breaches. Additionally, exposure of PII or payment data would likely trigger regulatory noncompliance and reputational damage.

Recommendation:

Immediately deploy a Web Application Firewall (WAF) with SQL injection protection and implement input rate limiting to block attack patterns. For a permanent fix, refactor code to use parameterized queries or ORM frameworks, and enforce strict input validation and sanitization across all endpoints. After remediation, rotate all credentials and tokens, revoke active sessions, and establish continuous security scanning (DAST/SAST) to detect and prevent future vulnerabilities.

Hardcoded / Leaked DB Credentials (user-data)

The EC2 user-data script used to provision the instance contains the MariaDB root password in plaintext (vulnerable) and executes SQL commands using that credential during initialization.

```
[ec2-user@ip-10-0-1-157 ~]$ sudo cat /var/lib/cloud/instances/*/user-data.txt
#!/bin/bash
LOG_FILE="/var/log/user_data.log"

# Update the instance
echo "Updating the instance" | tee -a $LOG_FILE
yum update -y 2>&1 | tee -a $LOG_FILE

# Install required software (Node.js, NPM, and MariaDB)
echo "Installing Node.js and MariaDB" | tee -a $LOG_FILE
curl -sL https://rpm.nodesource.com/setup_24.x | bash - 2>&1 | tee -a $LOG_FILE
yum install -y nodejs mariadb105-server.x86_64 git 2>&1 | tee -a $LOG_FILE

# Start and enable MariaDB
echo "Starting and enabling MariaDB" | tee -a $LOG_FILE
systemctl start mariadb 2>&1 | tee -a $LOG_FILE
systemctl enable mariadb 2>&1 | tee -a $LOG_FILE

# Set up MariaDB (vulnerable root password)
echo "Setting up MariaDB" | tee -a $LOG_FILE
mysql -u root -e "SET PASSWORD FOR root@'localhost' = PASSWORD('vulnerable');" 2>&1 | tee -a $LOG_FILE
mysql -u root -p'vulnerable' -e "CREATE DATABASE juice_shop;" 2>&1 | tee -a $LOG_FILE
```

Figure 16: Hardcoded DB Credentials

Impact:
Confidentiality & Integrity: Full DB compromise; data exfiltration; ability to create backdoor accounts or disable protections.

Recommendation:
Remove plaintext secrets from user-data. Use AWS Secrets Manager or SSM Parameter Store and grant the EC2 instance an IAM role with the secretsmanager:GetSecretValue permission. Adopt secret rotation and eliminate the use of root for application operations. Harden access to instance metadata and audit access logs.

Weak Database Password & Missing Secure Initialization

The MariaDB root account uses a weak static password and lacks secure initialization (no password policy, no privilege hardening).

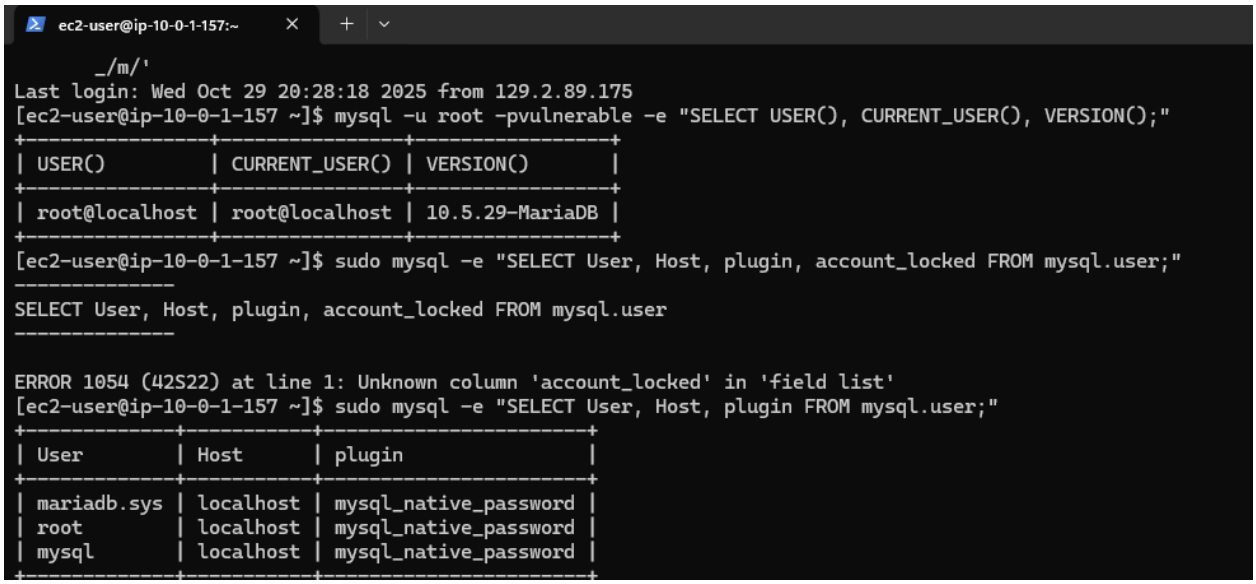

Figure 17: Privileged Accounts Using MySQL Native Passwords

Impact:
A weak root password combined with unrestricted privileges enables complete administrative control. An attacker can log in as root, create new privileged accounts, or disable security features such as auditing or SSL, leading to total data compromise.

Recommendations
Rotate the database root password to a strong, unique secret and remove it from any exposed locations. Secure the MySQL instance by removing anonymous users, disabling remote root access, and enforcing password validation. Create a least-privilege application account with only necessary permissions, restrict root access to localhost, and regularly audit user privileges to maintain database security.

No Encryption at Rest (EBS Volume Unencrypted)

The EC2 instance's EBS volume, which stores the database files, is not encrypted.

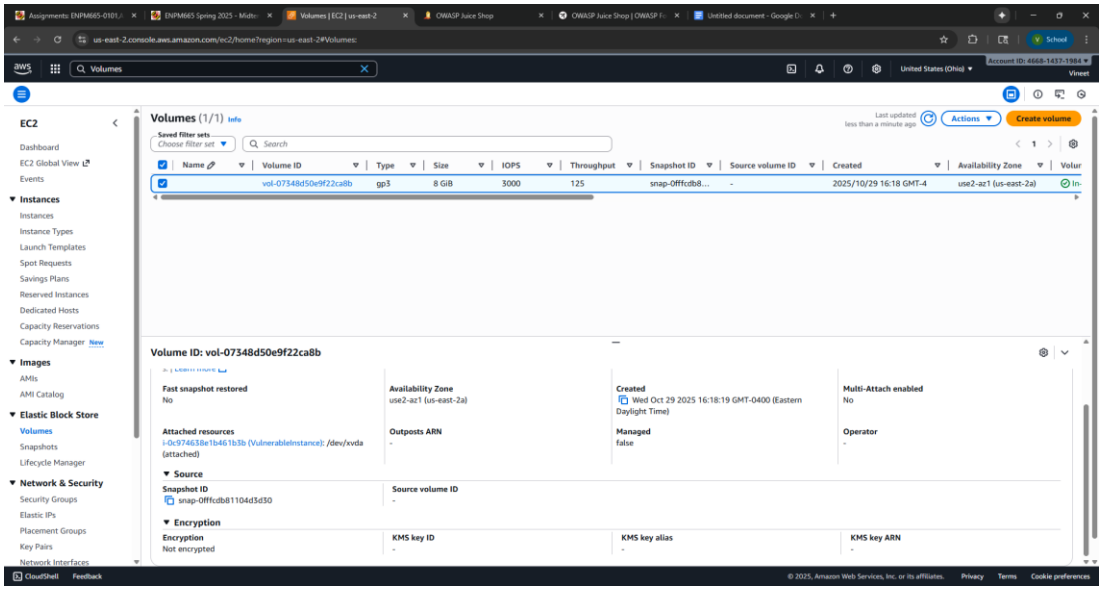

Figure 18: Unencrypted EBS Volumes

Impact:

Unencrypted volumes (snapshots, AMIs) can be accessed and read if a snapshot is copied or an attacker gains storage access

Recommendation:

Enable EBS encryption for all volumes and set encryption as the account default. Recreate or migrate existing data using encrypted snapshots, and ensure all backups and snapshots are encrypted and protected with strict access controls to maintain data confidentiality and compliance.

No Encryption in Transit (TLS/SSL Disabled)

MariaDB shows SSL/TLS is disabled (have_ssl = DISABLED), with no server certificate configured. Database connections, therefore, occur over plaintext.

```
[ec2-user@ip-10-0-1-157 ~]$ sudo mysql -e "SHOW GLOBAL VARIABLES LIKE 'have_ssl';"
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_ssl      | DISABLED |
+---------------+----------+
[ec2-user@ip-10-0-1-157 ~]$ sudo mysql -e "SHOW VARIABLES LIKE 'ssl%';"
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| ssl_ca        |       |
| ssl_capath    |       |
| ssl_cert      |       |
| ssl_cipher    |       |
| ssl_crl       |       |
| ssl_crlpath   |       |
| ssl_key       |       |
+---------------+-------+
[ec2-user@ip-10-0-1-157 ~]$ mysql -u root -pvulnerable -e "SHOW STATUS LIKE 'Ssl_cipher';"
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Ssl_cipher    |       |
+---------------+-------+
```

Figure 19: No Encryption in Transit (SSL Disabled)

Impact:
Unencrypted database connections expose credentials and sensitive query data to interception, leading to critical loss of confidentiality and potential unauthorized access.

Remediation:
Enable TLS/SSL for all database connections, enforce SSL requirements for database accounts, and use secure channels (TLS 1.2+) end-to-end; for managed services like AWS RDS, activate “Require SSL” and distribute the client CA for secure authentication.

Insecure Secrets Management

Application and system credentials are stored in plaintext across multiple configuration and code files within the /opt/juice-shop directory. Searches reveal embedded passwords, tokens, and sensitive environment variables inside YAML and JSON configuration files instead of a centralized secret store.

Figure 20: Application and System Credentials Stored in Plaintext

Impact:
Storing plaintext secrets in application files risks theft of database credentials and tokens, allowing attackers to alter configurations, inject malicious data, or disrupt services, resulting in high confidentiality, integrity, and availability risks.

Remediation:
Move all secrets to a secure store like AWS Secrets Manager or SSM Parameter Store, enforce minimal IAM access, remove plaintext secrets from code and configuration files, enable automatic rotation, and implement regular scans to prevent reintroduction.

Database Auditing & Logging Disabled

MariaDB has no auditing plugin installed, and both general and slow query logging are disabled; no log files were present for inspection.

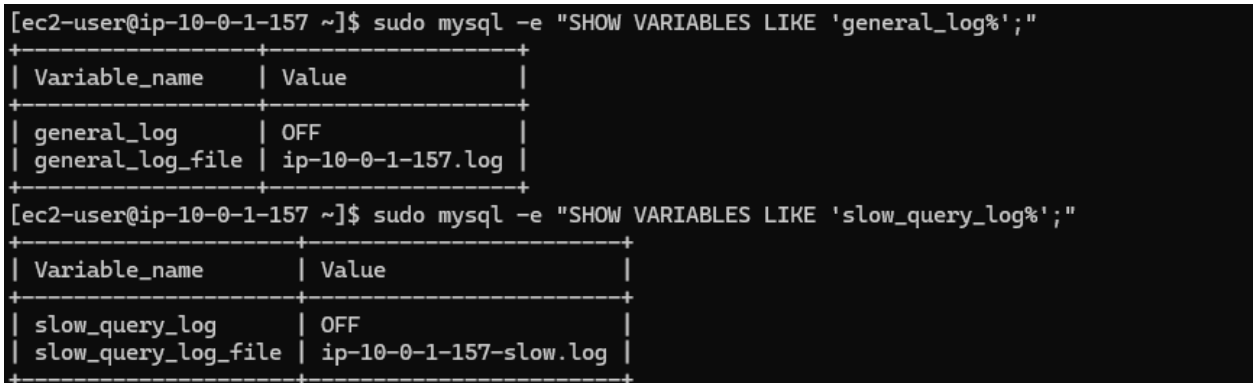

Figure 21: Database Auditing & Logging Disabled

Impact:
Lack of database logging severely limits the ability to detect suspicious activity or investigate incidents, reducing security monitoring and forensic capabilities.

Remediation:
Enable database auditing using MariaDB audit plugins or RDS Enhanced Auditing, activate general and slow query logs, forward logs to secure storage (CloudWatch/S3), and configure alerts for abnormal activity patterns.

Key Takeaways

The assessment showed several serious weaknesses that put sensitive user and system data at risk. Issues like SQL injection, exposed credentials, missing encryption, and weak passwords make it easy for attackers to steal or alter information. Storing secrets in plain text and not having proper logging also make it harder to detect or investigate security incidents. To fix this, the team should start by enabling encryption, using secure password and secret management tools, and setting up proper monitoring and audit logs. Strengthening these basic protections and following AWS and OWASP security best practices will make the system much safer, more reliable, and better prepared to handle real-world threats.

Virtual Machine Vulnerability Assessment

Introduction

This section focuses on vulnerabilities identified within the Amazon EC2 virtual machine hosting the e-commerce web application. The assessment used **AWS Inspector** to perform an automated scan of the EC2 instance to detect software vulnerabilities, insecure configurations, and missing patches. Manual verification was also conducted through system command checks to confirm Inspector's findings.

Vulnerabilities

The assessment identified several critical and high-severity vulnerabilities, including deprecated Tar package, Open ports, File system enumeration, XSS injection, and outdated OpenSSH version.

Tar Software Package

AWS Inspector found a flaw in node-tar 7.5.1 where reading a modified tar file could expose uninitialized memory data.

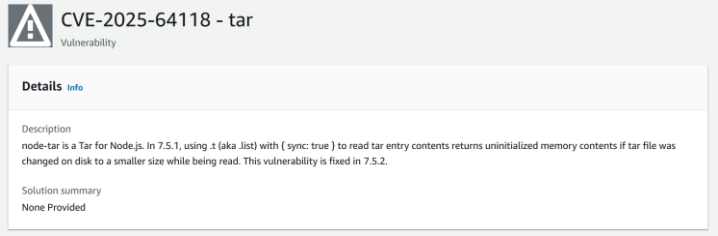

Figure 22: Security Flaw in Node-Tar 7.5.1

Impact:
An attacker could exploit this flaw to access sensitive memory contents or trigger application instability. In environments handling uploaded files or archives, this vulnerability could leak confidential data or be chained with other attacks for privilege escalation.

Remediation:
Update node-tar to version 7.5.2 or later, where the vulnerability is patched.

Publicly Accessible Ports

The EC2 instance has multiple publicly accessible ports. Port 22 (SSH) is open to the Internet, allowing external login attempts. Ports 80 (HTTP) and 443 (HTTPS) are also open but unused. This configuration increases the attack surface and exposes the instance to unnecessary risk.

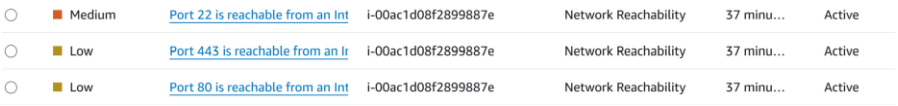

Figure 23: Publicly Accessible Ports

Figure 24: Port 22 listening

Impact:

Public SSH access enables brute-force and credential-stuffing attacks. Open but unused web ports can be exploited by automated scanners or misconfigurations in the future.

Remediation:

Restrict SSH (port 22) to trusted IPs only, or use AWS Systems Manager Session Manager. Remove or block inbound rules for ports 80 and 443 if no web service is required.

File system Enumeration

The Juice Shop installation exposes internal directories and files through its web root such that an attacker can enumerate or fetch sensitive artifacts by requesting known paths (ftp, config, profile, uploads, assets). The application directory listing shows multiple sensitive items present in the web root.

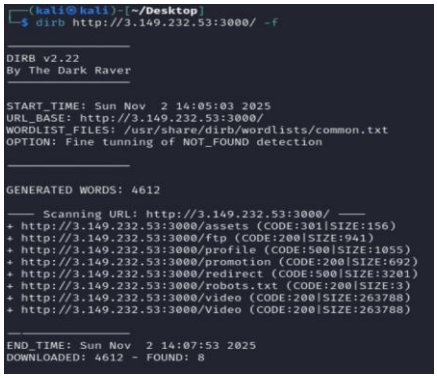

Figure 25: Exposed Internal Directories

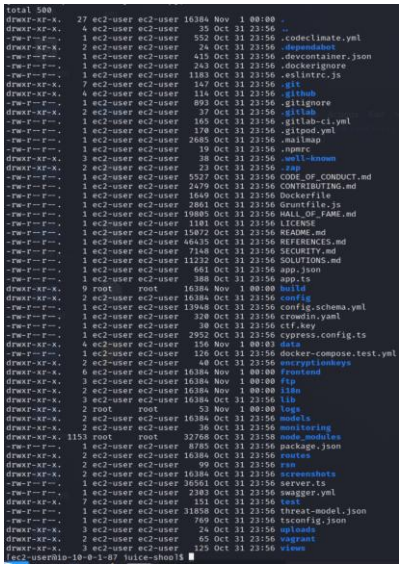

Figure 26: Exposed Internal Directories 2

Impact:
Disclosure of source code, configuration, encryption keys, application secrets, and internal documents. This enables credential harvesting, confidential data exposure, easier exploitation

Remediation:
Disable directory listing in the web server configuration, move sensitive files and folders outside the web root, Restrict access to internal directories like /ftp and /uploads. And apply proper file permissions (no public read or execute).

XXS injection

The application fails to properly sanitize user-supplied input before rendering it in the page context. This allows injection of arbitrary JavaScript code that executes in the user's browser, demonstrating a reflected Cross-Site Scripting (XSS) vulnerability.

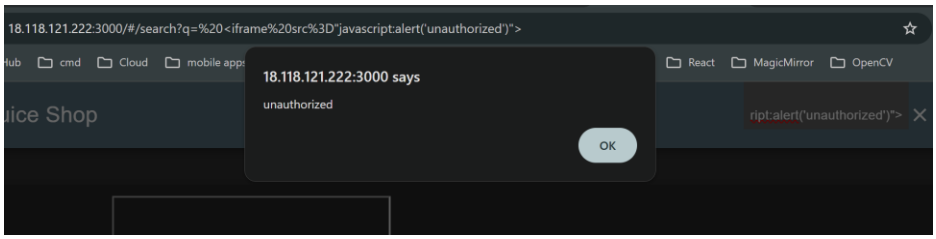

Figure 27: XSS Injection

Impact:
An attacker could exploit this vulnerability by executing arbitrary JavaScript code within the victim's browser. This could allow session hijacking, cookie theft, credential compromise, or phishing attacks. Because the malicious code executes in the context of the web application, it threatens the confidentiality and integrity of user sessions and data.

Remediation:
The application should validate and sanitize all user input on the server side and apply proper output encoding before rendering it in the browser. Developers should use secure templating frameworks that automatically escape user input and enforce safe rendering practices, and AWS WAF should be implemented to help detect and block malicious web requests, including cross-site scripting payloads and other injection attempts, before they reach the application.

Outdated OpenSSH Version

The assessed EC2 instance is running OpenSSH version **8.7p1** with OpenSSL **3.2.2**. OpenSSH 8.7p1 is older than current stable releases and is associated with several publicly disclosed vulnerabilities that can enable privilege escalation, user enumeration, remote code execution. Although the OpenSSL library on the host is up to date, the outdated OpenSSH binary increases the instance's attack surface

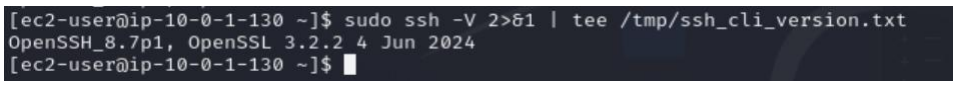

Figure 28: Outdated OpenSSH Version

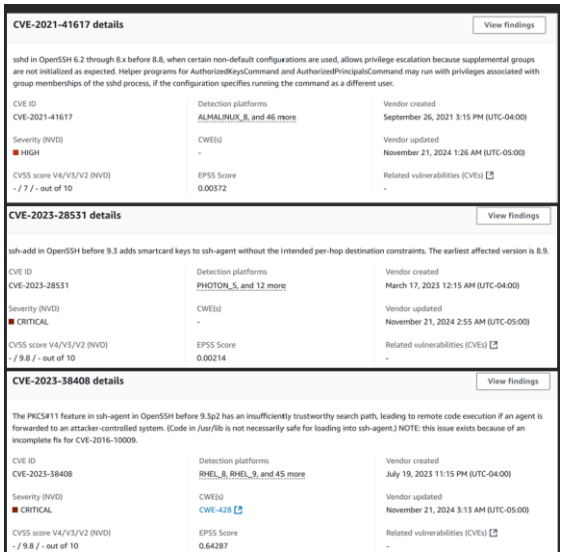

Figure 29: Vulnerabilities in Outdated OpenSSH Version

Impact:

An attacker who can reach SSH (port 22) could leverage these vulnerabilities (directly or indirectly) to escalate privileges, enumerate users, or if agent-forwarding is enabled. Execute code via malicious PKCS#11 modules. In a cloud environment this increases risk of lateral movement, credential compromise, or host takeover.

Remediation:

Upgrade OpenSSH to the latest stable release provided by the operating system vendor (≥ 9.3p2) to ensure all recent security patches are applied.

Key Takeaways

The EC2 assessment revealed outdated software, exposed ports, and insecure configurations that increase the attack surface. Public SSH and unused web ports allow potential unauthorized access, while directory listing and XSS flaws expose sensitive data. Updating software, restricting network access, and enforcing input validation will significantly improve the instance's security posture.

Network Security Assessment

Security Group Analysis

The initial Security Group (ENPM665GroupProject-WebServerSecurityGroup) allowed unrestricted inbound traffic on ports 22 (SSH), 80 (HTTP), 443 (HTTPS) from any IP (0.0.0.0/0). Outbound rules also permitted all traffic to all destinations, which increases the risk of data exfiltration.

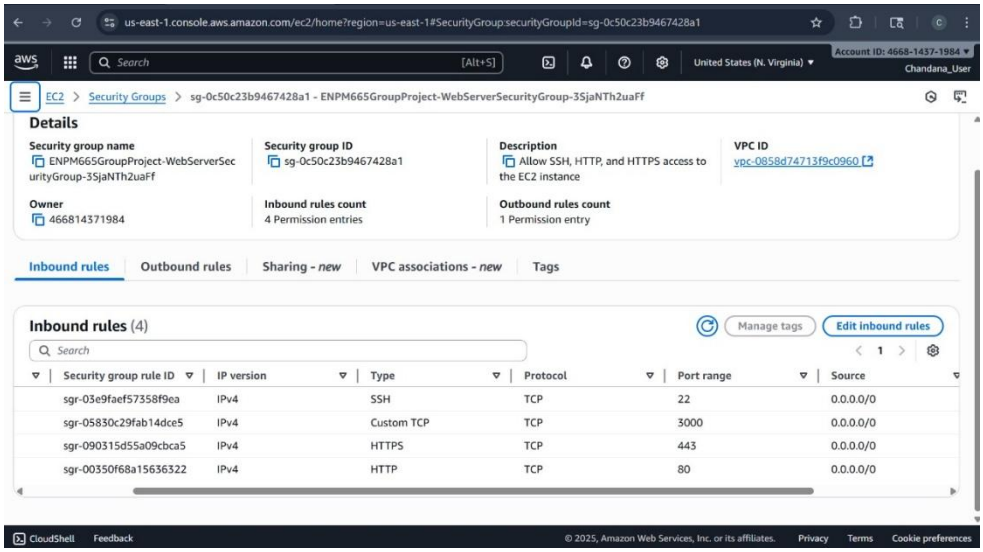

Figure 30: Security Group – Inbound Rules

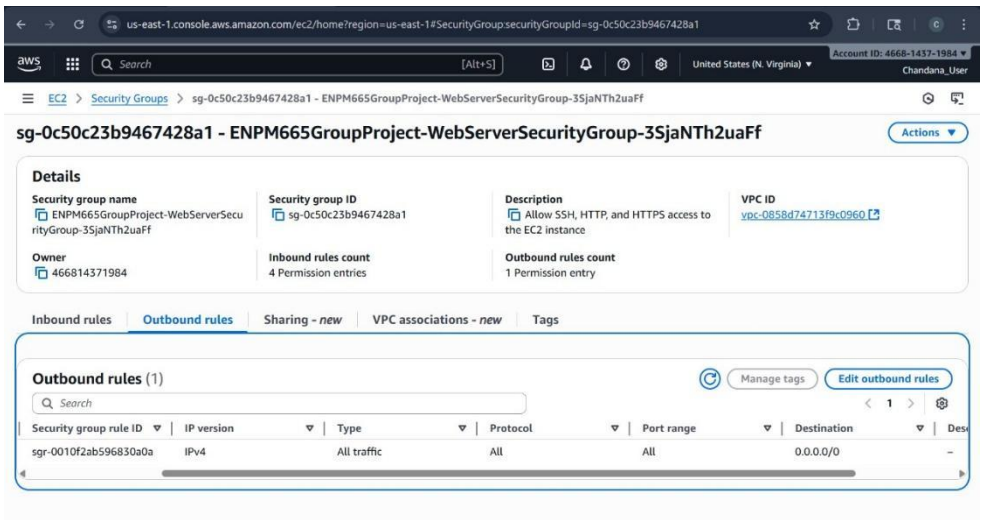

Figure 31: Security Group – Outbound Rules

This configuration exposes the EC2 instance to brute-force attacks (via SSH) and unencrypted web connections (via HTTP).

Impact:
Unrestricted network access exposes systems to potential attacks, including exploitation of open services and malicious requests.

Remediation:
Restrict public access by closing unused ports and enforcing least-privilege network rules.

Nmap port scanning and PowerShell connectivity tests were used to validate network exposure. These technologies verified that vital services like SSH (22) were open to the public.

Missing Network Monitoring and Flow Log Setup

Impact:
Without properly configured VPC Flow Logs, the organization lacks visibility into inbound and outbound network traffic, limiting the ability to detect anomalies, investigate incidents, or respond to potential security threats.

Remediation:
Ensure VPC Flow Logs are correctly configured with proper IAM permissions, send traffic data to CloudWatch Logs for both ACCEPT and REJECT events, and use monitoring and analysis tools to detect suspicious activity in near real-time.

Network Security Findings

Vulnerability	Evidence	Impact	Severity	Recommendation
Unrestricted SSH (Port 22)	Security Group screenshot	Enables remote brute-force access	High	Disable SSH from internet, use AWS Systems Manager (SSM)
Open HTTP Port (80)	Inbound Rules	Exposes unencrypted traffic	Medium	Redirect HTTP to HTTPS
All Outbound Traffic Allowed	Outbound Rules	Data exfiltration possible	Low	Limit outbound to required AWS services
Lack of Network Segmentation	VPC/Subnet config	Lateral movement possible within network	High	Create private subnets for backend components
No Initial Monitoring	Flow Log setup	No visibility into incoming threats	High	Maintain Flow Logs + integrate with AWS GuardDuty

Mitigation and Improvements

Segmentation

Implement a **multi-tier VPC** - Public Subnet for web layer, Private Subnet for DB and internal services.

Access Control

Restrict inbound rules to necessary IPs and ports. Remove public SSH and rely on AWS **Session Manager**.

Encryption

Configure an **Application Load Balancer (ALB)** with **ACM SSL certificate** to enforce HTTPS traffic.

Monitoring

Maintain VPC Flow Logs at 1-minute intervals and integrate **CloudWatch Alarms** or **GuardDuty** for intrusion alerts.

IAM Governance

Continue enforcing least-privilege IAM roles and monitor access via **AWS CloudTrail**

Key Takeaways

The network security assessment focused on evaluating how well the e-commerce application's cloud infrastructure was protected from unauthorized access. We reviewed the VPC setup, security groups, and IAM roles to identify misconfigurations and potential exposure points. Using PowerShell and Nmap, we verified which ports were open and tested access controls. Port 3000 was intentionally opened for controlled application access and therefore was not treated as a vulnerability. Based on our findings, we recommended restricting public access to only necessary ports, enforcing least-privilege IAM policies, and enabling continuous monitoring through VPC Flow Logs. These recommendations aim to reduce attack surfaces, improve visibility, and align the environment with AWS best practices.

Logging and Monitoring Analysis

Summary

The CloudFormation from this .yaml file creates a publicly reachable EC2 instance where an e-commerce platform "Juice Shop" is hosted. This cloud infrastructure is missing critical logging and monitoring functionalities like 'centralized logging', 'Threat Detection', and 'AWS Management Auditing'. This limits the visibility of the entire infrastructure. The ability to detect any cyberthreats, investigating security incidents, monitoring of lateral movement, data exfiltration, or insider threats is very limited. This significantly increases difficulties in security operations and poses high regulatory/compliance risk.

Current Visibility Summary

1. Local logging to /var/log/user_data.log – Log data will be lost if instance is terminated.
2. No CloudTrail for auditing activity – No visibility for AWS API activity.
3. No CloudWatch Logs defined – No automatic tracking and alerting for metrics like CPU and memory usage.
4. No VPC Flow logs – Missing visibility to network traffic information for the infrastructure.
5. No GuardDuty, Security Hub, Detective, or AWS Config – Missing out on 'threat detection capabilities', 'aggregation for security findings', and 'resource configurations and compliance tracking'.
6. No Centralized Logging – No provisions for centralized collection of logs for monitoring.

Primary Remediations

1. Enable AWS CloudTrail for both data and management related events
2. Enable VPC Flow Logs, GuardDuty, and AWS Config across the infrastructure. Centralize the logs using dedicated S3 bucket and accounts.
3. Deploy CloudWatch Logs and install CloudWatch agent on the EC2 instance.
4. Configure log metric filters and alarms for critical signals. Monitor Root account usage, sign-in without MFA, and security group changes.

Key gaps in detection & response

1. Missing account-level audit (CloudTrail)

Technical gap: AWS API activity is not recorded.

Risk: High chance of unauthorized changes going unnoticed, this enables malicious actors to damage the infrastructure without leaving trace, and sustain stealthy persistence. CloudTrail and Alarms for critical is required according to CIS benchmark.

2. No Centralized Log Collection (CloudWatch Logs / SIEM)

Technical gap: Logs are stored locally on EC2 instance.

Risk: The log data will be lost if the EC2 instance is terminated. Logs can be easily deleted or tampered if instance is compromised. Building organization level dashboards or alarms is not possible.

3. No Traffic Monitoring (VPC Flow Logs)

Technical gap: No flow logs to record accepted/rejected traffic at the subnet or ENI level.

Risk: Lateral movement, data exfiltration, or anomalous scanning will be invisible at network level—critical for detecting stealthy attackers.

4. No Threat Detection (GuardDuty) / Automated Enrichment (Detective)

Technical gap: No managed threat detection or investigation automation.

Risk: Activity related to known Indicators of compromise (IOCs) will not be visible, increasing chances of a security breach.

5. No Immutable Storage, No Retention Policy

Technical gap: Logs are not stored in safe storage.

Risk: Logs can be tampered or deleted if the instance is compromised. This poses high risk of compliance breach.

6. No Alerting or Playbooks

Technical gap: No CloudWatch alarms, or runbooks.

Risk: Chances of security events going unnoticed, even if incidents are detected no predefined runbooks means high mean time to respond (MTTR) and increased business impact.

Recommended Remediations

Quick Actions

1. Enable AWS CloudTrail – Enable multi-region AWS CloudTrail to store logs in dedicated, secure, centralized S3 bucket.
2. Setup CloudWatch – Setup basic metric filter to alarm metrics like CPU and memory usage.
3. Enable VPC Flow Logs – Collect network traffic logs to secure, centralized S3 bucket.
4. Install CloudWatch Agent on EC2 – Setup agent to push local logs from EC2 instance to CloudWatch logs.
5. Enable Amazon GuardDuty – Setup Amazon GuardDuty to monitor malicious activity.
6. S3 Object Lock / MFA Delete – Use Object lock and MFA Delete for the S3 buckets used for logging purposes.

Next Steps

1. SIEM Integration – Deploy SIEM tool for advanced correlation, threat hunting, and long-term retention.
2. Enable AWS Config – Setup AWS Config to monitor security group changes, IAM Policy changes.

3. Deploy Amazon Detective – Setup Amazon Detective alerting for known Indicators of compromise (IOCs).
4. Playbook Creation – Create playbooks for high impact security incidents, test these playbooks by simulating the incidents.

Long Term Plans

1. SOC Automation (SOAR) – Automate the playbooks for quick response and remediations.
2. KPI Monitoring – Monitor and improve performance KPIs.

Key Performance Indicators (KPIs) to Monitor

1. MTTD (Mean Time to Detect) – Track and improve MTTD. Target < 15 minutes for critical incidents
2. MTTR (Mean Time to Remediate) – Confirmed breached should be contained within 4 hours.
3. Log Retention – Percent of security logs retained for compliance purposes. Schedule recurring backups for logging buckets. Target – 13 months for usual compliances.
4. Meeting SLA (Service level agreements) – Aim towards 100% SLA compliance.

Cost & resource considerations

Costs of the recommended security control depend upon the log volume, retention period, number of regions. The major chunk of cost and resource allocation will be required for SIEM integration and log ingestion. AWS features like CloudTrail, Amazon Detective, AWS Config, VPC Flow logs require much lesser cost and resource allocation. These features provide comparatively higher ROI. Centralized S3 storage costs depend upon memory used, retention period and access frequency. Using less active storage types like AWS Glacier can help reduce cost drastically for storing log archives. CloudWatch Logs and GuardDuty require moderate cost considerations. However, these costs are much lesser than costs breach remediation and regulatory fines.

Key Takeaways

Enable AWS CloudTrail, VPC Flow Logs, and Amazon GuardDuty on priority. This will enable basic visibility into the infrastructure. Then move to Centralized logging, SIEM Integration, AWS Config, Amazon Detective, and Playbook Creation. Which will help to keep eagle's eye view on entire infrastructure and quickly responds to threats. Finally create streamline process of integrating SOAR (Security Orchestration, Automation, and Response) capabilities, monitoring, and improving the KPIs. This will create a high visibility environment which will drastically help in quick detection and swift remediation of the security incidents.

Disaster Recovery Review

Introduction

This assessment identifies critical risks that could result in extended downtime or permanent data loss. A tested disaster recovery (DR) plan with specified Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO) should be established, along with automated, encrypted backups, AMI (Amazon Machine Image) and EBS (Elastic Block Store) snapshot creation, and credential security using AWS Secrets Manager.

Current Disaster Recovery (DR) Posture:

The current deployment lacks redundancy and automation data protections.

- Single Point of Failure: Only one EC2 instance exists in a single Availability Zone.
- Local Database Storage: MariaDB and application files reside on the EC2 root volume without scheduled backups.
- No Automated Backups or Snapshots: The deployment does not include AMIs, EBS snapshots, or AWS Backup configurations.
- Exposed Security Risks: The instance allows SSH, HTTP, HTTPS, and port 3000 access from all IP addresses, with weak database credentials.
- No DR (Disaster Recovery) Documentation: There is no documented recovery procedure or testing schedule.

The absence of above elements significantly increases the risk of data loss and extended downtime.

Risk and Potential Impact:

Risk	Cause	Potential Impact	Severity
Single EC2 instance	Single AZ (Availability zone) deployment	Instance failure or AZ outage causes downtime	High
Local database	MariaDB stored on instance	Data loss if the instance is terminated or the disk fails	Critical
No snapshots or AMIs	No automated image creation	Manual rebuild required, extended downtime	High

Weak security	Open ports and weak DB password	Data corruption or deletion by attackers	High
No documented RTO/RPO	Lack of DR plan	Inconsistent or delayed recovery	Medium

Disaster Recovery Objectives:

Objective	Target	Justification
RTO (Recovery Time Objective)	**<=1 hour**	Downtime longer than an hour can result in a substantial loss of revenue and unhappy customers for a live e-commerce site. Complete restoration within 60 minutes is possible with AWS infrastructure (auto scaling, multi-AZ failover, and snapshots).
RPO (Recovery Point Objective)	**<= 15 minutes**	Payment information, customer orders, and transactions must all be preserved. This data protection goal is doable and safe for businesses with automated backups every 5 to 15 minutes and database replication (like Amazon RDS Multi-AZ or Aurora).

Recommended Backup and Recovery Strategy:

Immediate Actions:

- Create initial AMI and EBS snapshot to capture the current system state.
- Export the database via MySQL dump and store encrypted in S3 with versioning enabled.
- Secure credentials and network: restrict SSH to trusted IPs and store database passwords in AWS Secrets Manager.

Short-Term Automation:

- Automate database dumps to S3 with cron jobs.
- Schedule regular EBS snapshots using AWS Data Lifecycle Manager or AWS Backup.
- Isolate backups: store backups in a separate account or region to protect against compromise.

Mid-Term Architectural Improvements:

- Migrate database to RDS/Aurora for Multi-AZ redundancy and automated backups.
- Implement multi-AZ deployment for EC2 and consider cross-region replication for backups.
- Use immutable infrastructure with AMI-based deployments and Auto Scaling for quick recovery.

Recovery Procedure (Runbook):

Restore from AMI

- Launch new EC2 instance from AMI.
- Associate existing Elastic IP or update DNS.
- Verify application and database connectivity.
- If database is not included in AMI, restore from latest S3 backup.

Restore from EBS Snapshot

- Create new EBS volume from snapshot.
- Attach to new EC2 instance and mount or copy data.
- Start services and validate operation.

Restore from RDS Snapshot (if DB migrated)

- Use RDS snapshot to restore the database in multi-AZ configuration.
- Update application connection string to restored database.

DR Governance, Testing, and Maintenance

- **Runbook Documentation:** Include backup location, responsible personnel, and step-by-step recovery instructions.
- **Testing Schedule:**
    - Weekly: DB restore to test instance.
    - Monthly: Full recovery drill.
    - Quarterly: Failover test for cross-region backups.
- **Success Criteria:** Verify RTO and RPO targets are met, and application is fully functional.

Additional Recommendations:

- Restrict public access to SSH and application ports.
- Replace weak database credentials and manage them securely.
- Consider using a private subnet with a bastion host or AWS Session Manager for secure management.
- Enable CloudWatch monitoring and alerts for system health.
- Implement automated lifecycle management for snapshots and backups to control costs.

Prioritized Action Checklist:

Priority	Action	Timeline
1	Take initial AMI/EBS snapshot, MySQL dump to S3, secure DB password	Immediate
2	Automate backups via cron and IAM role	Within 48 hours
3	Schedule AWS Backup or DLM snapshots	Within 1 week
4	Migrate DB to RDS Multi-AZ and implement AMI deployment pipeline	Within 2–4 weeks
5	Document runbook and conduct regular DR tests	Ongoing

Key Takeaways

The disaster recovery assessment emphasizes the critical importance of safeguarding vital data and maintaining business continuity. The current single-EC2 instance and single-AZ configuration create potential points of failure, making robust recovery strategies essential. Data protection is enhanced by using MariaDB-dump backups stored in S3 with encryption and versioning. At the same time, AMIs and EBS snapshots enable quick instance-level recovery in the event of a system failure. Resilience and availability are further improved by implementing multi-AZ deployments and managed databases such as RDS. Automated backups and scheduled snapshots ensure that the RPO of ≤15 minutes and RTO of ≤1 hour can be achieved, reducing the likelihood of business interruption, data loss, or outages. These measures also help uphold customer trust and protect sensitive client information. Overall, a well-organized disaster recovery plan guarantees that the company can recover rapidly from setbacks, providing a robust, secure, and scalable framework for production operations.

Conclusion

The overall security assessment revealed multiple areas of concern across identity and access management, application vulnerabilities, data security, virtual machines, and network configurations. Exposed emails, weak passwords, publicly accessible admin consoles, and SQL injection risks highlight weaknesses in access control and authentication. Additionally, missing encryption, hardcoded credentials, unmonitored logs, and insecure configurations on EC2 instances increase the potential for unauthorized access and data compromise. Strengthening password policies, moving sensitive data to private infrastructure, enforcing least-privilege IAM roles, and implementing secure secret management will significantly reduce these risks. Regular updates, patching, and proper input validation are also critical to maintain a resilient application environment.

On the infrastructure side, the assessment identified network misconfigurations, unrestricted port access, and the absence of continuous monitoring as key vulnerabilities. Implementing VPC segmentation, restricting public access to only necessary ports, enabling VPC Flow Logs, CloudTrail, GuardDuty, and integrating centralized logging with SIEM and SOAR capabilities will improve visibility and response to threats. Furthermore, disaster recovery measures, including multi-AZ deployments, encrypted backups, and automated snapshots, ensure business continuity and data protection. By addressing these findings holistically, the organization can achieve a more secure, reliable, and resilient system that protects both users and organizational operations.